CS435 Homework Assignment 5

Due date: ________________

Turn in all code on both paper and by email (to dbahr@regis.edu with “CS435 Homework” in the subject line).

Please include comments in all of your code. Describe what your function does, and describe anything that is tricky or unclear.

Problem #1: Write a simple Java method that moves all of the contents of a stack onto a queue. Use only the standard operations that apply to each ADT. (In other words, use push, pop, etc. for a stack. Use enqueue, dequeue, etc. for a queue.)

Your method does not need to run any other code and it does not need to exist inside of a larger program. In other words, assume that your method is passed a stack and a queue. Do everything you need to do inside this method. An example is given below.

Problem #2: Write a simple method that moves all of the contents of a queue onto a stack. Use only the standard operations that apply to each ADT.

Problem #3: Write a simple method that moves all of the contents of one stack onto another stack without changing the order.

You may use the methods created in problems 1 and 2 if they help. You may also use the standard operations of any of the ADTs we have studied thus far (lists, queues, stacks). Hint: Creating a temporary queue or stack might help.

Problem #4: Start with a queue and an empty stack. Write a simple method that uses the stack to reverse the order of the entries on the queue. (Your final answer should be stored on the queue.)

You may use the methods created in problems 1, 2, and 3 if they help. You may also use the standard operations of any of the ADTs we have studied thus far (lists, queues, stacks).
**Example Problem:** Write a simple method that moves all of the contents of a linked list onto a queue. Use the standard linked list operations of find, next, previous, insert, delete, findKth, first, etc. Use the standard queue operations of enqueue, dequeue, etc.

**Solution:**

```java
/*
 * The following code moves a linked list onto a queue. The method
 * takes two parameters, a linked list and a queue. The values of each
 * are assumed to be integers.
 */
public void moveListToQueue(LinkedList list, Queue q)
{
    //Get the first element of the linked list. Note that
    //the function “findKth” requires a position and a list. I give
    //it 0 since that’s the first position.
    ListNode nextNode = list.findKth(0);

    //One by one, grab each node from the list and
    //add it onto queue. When we run out of elements on the
    //linked list, nextNode will return a null. So I end the loop
    //when it detects a null.
    while(nextNode != null)
    {
        int nodeValue = nextNode.value;
        q.enqueue(nodeValue);
        nextNode = nextNode.next;
    }
}
```

Notice how I take advantage of the ADT operations of “enqueue” and “findKth” (I could also have used “first” instead of “findKth”). I also take advantage of the class called ListNode that was defined in the classroom. Your code for the above problems should have a similar style and structure. If you want to run and test your code, you can (1) type in the code that I gave you as handouts for the LinkedList, etc. Or you can use the LinkedList, and Stack from the Java API, but note that their implementation differs slightly from what I showed in class. But you have learned enough to see and understand the differences! You can also use Java’s LinkedList as a Queue. But if you find that a bit
annoying (I do), and you don’t want to transcribe the handout (what a pain), here’s my own implementation of a Queue. Feel free to copy and use this code.

```java
import java.util.LinkedList;

/**
 * A classic queue data structure. Elements are treated as FIFO (first-in-first-out).
 * Students of Introduction to Programming should be able to use this code.
 * Students of Object-Oriented Programming should be able to understand
 * the basic ideas behind this code. Students of Software Engineering, will
 * recognize that this class is built using the Adapter design pattern (adapting
 * the LinkedList). If you would like to completely understand this code, you
 * will also need to read about "generics" which allow the use of the <E>
 * notation.
 *<p>
 * This code was developed in 2009 for classroom use at Regis University. Any
 * other use is prohibited without permission from the faculty member and
 * author, David Bahr.
 */

public class Queue<E> {
    // this code adapts a linked list to use as a queue
    private LinkedList<E> list = null;

    /**
     * Creates an empty queue.
     */
    public Queue() {
        list = new LinkedList<E>();
    }

    /**
     * Removes and returns the first element of the queue.
     * @return <E> The first element, or null if the list is empty.
     */
    public E dequeue() {
        return list.poll();
    }

    /**
     * Inserts an element at the end of the queue.
     * @param <E> The element that will be inserted.
     */
    public void enqueue(E e) {
        list.add(e);
    }
}
/**
 * Returns true if the queue is empty.
 * @return true if the queue is empty.
 */
public boolean isEmpty()
{
    return list.isEmpty();
}

/**
 * Empties the queue by removing all elements.
 */
public void makeEmpty()
{
    list.removeAll(list);
}